Trong dự án đặt lịch khám bệnh cho bệnh nhân em sử dụng những công nghệ sau :

* Kafka dùng để giao tiếp giữa Appointment Service và Notification Service

( vì trong trường hợp message bị delay cũng không ảnh hưởng đến luồng business)

Cách sử dụng : phía bên Appointment sevice sẽ khai báo gửi mess ở producer bằng class KafkaTemplate ; dùng phương thức send()

Bên phía Notification sẽ sử dụng annotation @KafkaListener(topic =” ”,groupId = “ “) để khai báo consumer và nhận mess từ toppic

* Non - cluster index dùng để đánh vào cột email (unique) của bảng User , bảng doctor và bảng staff ( để người dùng đăng nhập trả về kết quả nhanh hơn ) – dùng trong trường hợp dữ liệu lớn

Cách sử dụng :  
CREATE INDEX idx\_email ON Users(Email);

Tạo 1 vùng nhớ với index được đánh vào cột email

SELECT \*

FROM Users

WHERE Email = 'bob@example.com';

* Partition được dùng chỉ khi dữ liệu trong 1 bảng quá lớn hoặc muốn quản lí lượng lịch đặt theo thời gian vì tác dụng là cải thiện hiệu suất truy vấn ( ví dụ mỗi năm tạo 1 partition mới hoặc chia patition cho 1 bảng đã có hàng triệu dữ liệu )

Cách sử dụng , chia bảng order theo năm :

CREATE TABLE Orders (

OrderID INT NOT NULL,

CustomerName VARCHAR(100),

OrderAmount DECIMAL(10, 2),

OrderDate DATE NOT NULL,

PRIMARY KEY (OrderID, OrderDate)

)

PARTITION BY RANGE (YEAR(OrderDate)) (

PARTITION p0 VALUES LESS THAN (2020),

PARTITION p1 VALUES LESS THAN (2021),

PARTITION p2 VALUES LESS THAN (2022),

PARTITION p3 VALUES LESS THAN (2023),

PARTITION p4 VALUES LESS THAN MAXVALUE

);

-- Lấy đơn hàng trong năm 2021

SELECT \* FROM Orders WHERE OrderDate >= '2021-01-01' AND OrderDate < '2022-01-01';

* Authen và Author bắt buộc phải dùng để đảm bảo hệ thống có sự bảo mật cao

Authentication là quá trình xác thực có thể hiểu theo câu hỏi ( bạn là ai )

Ví dụ trong quá trình login bằng tài khoản mật khẩu , sinh trắc học , vân tay ...

Authorization là quá trình phân quyền, có thể hiểu theo câu hỏi ( bạn được quyền làm gì ) quá trình này chỉ xảy ra khi và chỉ khi quá trình authentication thành công !

* JWT thuộc cả authen và author nên bắt buộc phải dùng

JWT là 1 chuỗi kí tự chứa các thông tin về xác thực và phân quyền trong phần thân ( payload ) ví dụ như customerId ( thông tin người dùng sau khi xác thực ) , role (quyền của customerId sau khi đã xác thực )

Phần chữ kí rất quan trọng của chuỗi jwt vì nó chứa secretkey ( không thể làm giả bởi úng dụng khác và Hệ thống có thể xác minh JWT được tạo bởi máy chủ của chính nó. )

Sau khi gửi request chứa jwt thì hệ thống sẽ lấy ra được role trong jwt và đối chiếu với api đã được setup role được truy cập xem có khớp với nhau không (đây là quá trình phân quyền ) nếu khớp thì nó sẽ cho truy cập đến api đó của hệ thống và thực hiện khối lệnh trong api

* Bean : hay dùng Singleton(chỉ 1 bean của 1 class khi tạo ra) và Prototype (khi gọi mới khởi tạo – dùng xong sẽ remove)
* @Transaction : dùng cho từng microservice nghiệp vụ , quản lí transaction cho từng microservice – đánh annotation cho hàm create ( appointment , payment ) những hàm này là những hàm tạo dữ liệu vào database dùng @Transaction để bắt exeption và remove những dữ liệu đã thêm vào database nếu transaction không success hết
* Microservice : là kiến trúc để phát triển dự án. Chia các microservice theo nghiệp vụ cụ thể ( tùy từng dự án sẽ chia khác nhau ) nhưng hầu hết các dự án đều có orderService và payment service
* Redis : trong project của mình , tiến hành cahe bác sĩ ( vì dữ liệu bác sĩ ít thêm sửa xóa ) – tận dụng điểm mạnh của redis là đọc dữ liệu nhanh

Ví dụ : @Cacheable(value = "patientsAll")

public List<Patients> findAllPatients() {

System.out.println("Fetching patients from database...");

return patientRepository.findAll();

}

Sau mỗi 1 ngày cache lại ( xóa cache cũ và gọi lại hàm trên ):  
@CacheEvict(value = "patientsAll", allEntries = true) // Xóa toàn bộ cache

@Scheduled(cron = "0 0 0 \* \* ?") // Chạy vào 0:00 mỗi ngày

public void refreshPatientsAllCache() {

System.out.println("Refreshing cache for patientsAll...");

findAllPatients(); // Gọi lại để làm mới cache }

Nếu có dữ liệu thay đổi , sẽ thay đổi giá trị của dữ liệu đó :  
@CachePut(value = "patients", key = "#patient.patientEmail")

public Patients changePassword(ChangePasswordRequest changePasswordRequest) {

Patients patient = patientRepository.findByPatientEmail(changePasswordRequest.getEmail());

if (patient == null) return null;

if (!passwordEncoder.matches(changePasswordRequest.getOldPassword(), patient.getPatientPassword())) return null;

patient.setPatientPassword(passwordEncoder.encode(changePasswordRequest.getNewPassword()));

return patientRepository.save(patient);

}

* Swagger : xuất bản api

Cách dùng :

Thêm dependences:  
<dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
 <version>2.6.0</version>  
</dependency>

Cài đặt trong file property :

springdoc.swagger-ui.path=/swagger-ui.html  
springdoc.api-docs.path=/v3/api-docs

Nếu microservice có security và có jwt thì phải cấu hình cho nó thêm jwt vào trong trang:

@OpenAPIDefinition(  
 security = @SecurityRequirement(name = "bearerAuth")  
)  
@SecurityScheme(  
 name = "bearerAuth",  
 type = SecuritySchemeType.*HTTP*,  
 scheme = "bearer",  
 bearerFormat = "JWT" // Định nghĩa JWT  
)  
public class SwaggerConfig {  
}

* Api get way : API Gateway là một thành phần trung gian giữa client (người dùng, ứng dụng) và các dịch vụ backend (microservices). Nó đóng vai trò như **điểm vào duy nhất** (entry point) cho toàn bộ hệ thống.

Chức năng chính :  
 **Định tuyến yêu cầu (Routing):**

* Chuyển yêu cầu từ client tới các dịch vụ backend dựa trên đường dẫn URL, phương thức HTTP, hoặc các quy tắc khác.

 **Bảo mật:**

* Xác thực và phân quyền (thường sử dụng JWT, OAuth2...).
* Kiểm soát tốc độ truy cập (Rate Limiting) để bảo vệ hệ thống.

 **Tích hợp:**

* Tổng hợp dữ liệu từ nhiều microservices và trả về kết quả hợp nhất cho client.

 **Quản lý lỗi (Fault Tolerance):**

* Hỗ trợ **circuit breaker** để ngăn ngừa lỗi lan truyền.
* Cung cấp fallback trong trường hợp microservice gặp sự cố.

 **Chuyển đổi dữ liệu (Transformation):**

* Chuyển đổi định dạng dữ liệu hoặc giao thức (HTTP sang gRPC, JSON sang XML...).
* Discovery Service là một thành phần trong hệ thống microservices giúp tự động **đăng ký và tìm kiếm** các dịch vụ (service) đang hoạt động. Nó đảm bảo các dịch vụ có thể giao tiếp với nhau một cách hiệu quả mà không cần cấu hình tĩnh.
* Thread – mutilthread – executorService:   
  Dùng khi thực hiện 1 công việc đơn lẻ mà không phải quản lí phứcc tạp ( ví dụ dùng trong trường hợp gửi email khi bệnh nhân đặt lịch xong )